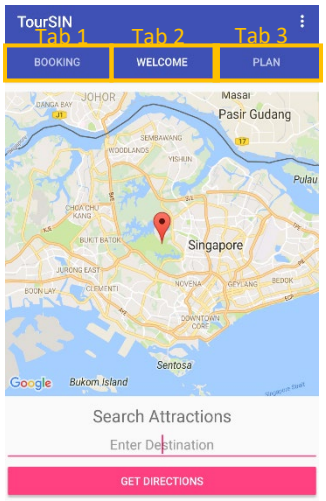
TourSIN

TourSIN is a travel app that assist tourists in locating popular attractions, booking of attraction tickets and hotel rooms, and itinerary planning.

App Features:

1. Overview

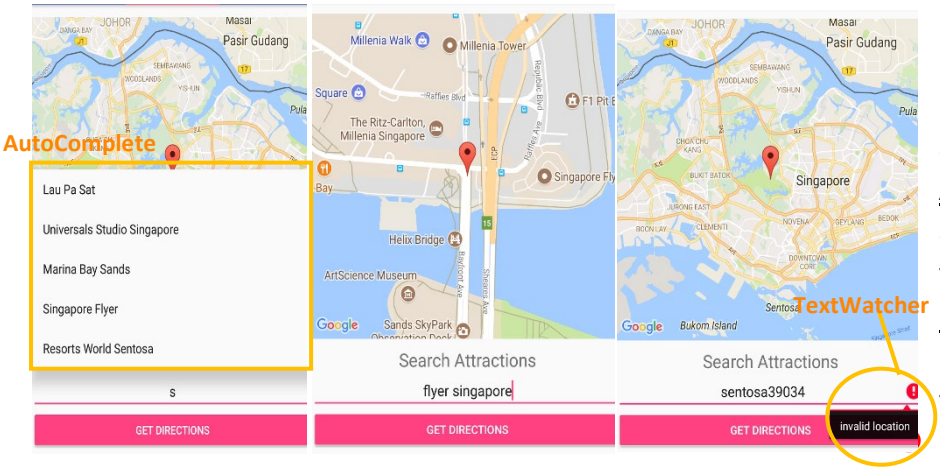


Upon launching the app, this is the default tab that displays the welcome screen, which shows a marker on Singapore.

With the Tab Layout implemented, users may choose from 3 different tabs, to book attraction ticket and hotel rooms, to search for local attractions, and to plan the specific route for their itinerary.

The user may tap on each tab header to move to that tab, or swipe left and right to navigate between tabs. The currently active tab would have its tab header lighted up.

2. Attractions Locator



Users may search for attractions by entering the location into the search bar. With the aid of the AutoComplete feature, the app suggests possible attractions to the users. It also saves the search history and include past searches into the list of suggestions from the Autocomplete feature.

Upon clicking “Get Direction”, the Google Map fragment shows a MapView of the location marker of the searched location, as shown in the second figure above.

Multiple measures were set-in place to ensure that the system is robust to typos, including:

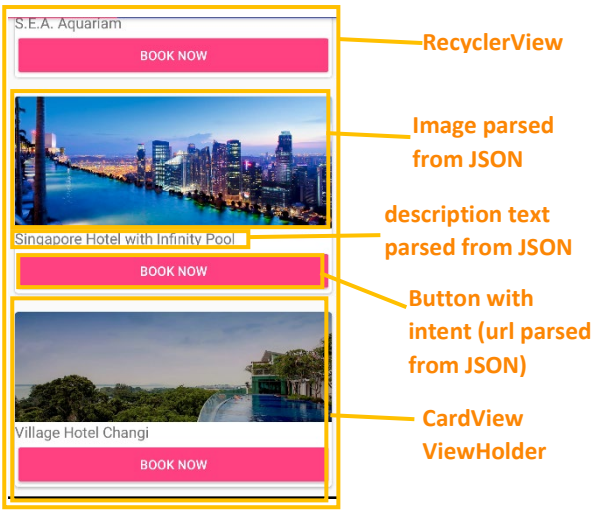
- Google Map’s built-in Autocorrect
- **FuzzyCheckerFrej.java** implements the open-source Java Library Fuzzy Regular Expression for Java (FREJ), this ensures that the search bar is more resistant to user typos.
  E.g. User input: “flyer singapore”, it will still lead them to the right destination Singapore Flyer.
- **ValidationRegex.java** implements regular expressions such as “[A-Z,a-z]+[0-9]+” to filter the types of possible location errors that users may input.
- TextWatcher is also implemented to verify and send alerts to user if they have entered an invalid location, or if no text was entered.

3. Booking Attraction Tickets and Hotel Rooms

The Booking Tab allows users to book attraction tickets and hotel rooms, often at a discounted price. Each booking is included in a CardView, and the CardView ViewHolder objects are arranged in a RecyclerView. Inside each CardView ViewHolder, there is an Image of the attraction/hotel, a text description of the place, and a “Book Now” button. The image file, text description string and the booking url of each place are saved in and retrieved from a JSON file.

This allows us to efficiently use the ViewHolder pattern to generate each CardView object from the JSON file.

The button calls an implicit intent that opens the url in a browser, which leads to the respective booking pages of each attraction or hotel.



In summary, the features used in Function 1 are:

- Tab Layout
- Function 2
- Autocomplete
- Search History
- MapView
- Implicit Intent
- FuzzyChecker
- ValidationRegex
- TextWatcher
- CardView
- RecyclerView
- JSON

4. Itinerary Planning

Algorithm Analysis

We assume that the travelling time between all locations is proportionate in all modes of transport.

Brute force method

All possible combinations to travel will be generated, with the start point remaining fixed. From there we will run through every single combination generated, to get the fastest route via taxi. Once the fastest route is given, we will consider the price of the fastest method. If it is higher than the budget given, we will change the transport method used for the particular edge. We will try to switch from taxi to public transport for each edge, and if an all public transport route is still unable to give us the desired cost, we will consider to switch from public transport to walking for each edge.

Algorithmic method

For the algorithmic approach, we will use the same assumption established in the brute force method. We will use a greedy approach to determine where to go next. A sorted list sorted by the time needed to travel by taxi is provided as a data. The algorithm decides where to go next by determining based on the current node, which unvisited node is the closest. The final location will go back to the start point. The same approach to reduce cost is used here. This is ideal as it cuts down on the distance that one needs to travel from one node to another.

Given all permutations is considered, the complexity will be O(n!) with n being the number of places that the user intends to visit. This is due to the large number of combinations that needs to be considered. As such, if the user decides to visit a large number of places, it may not be very advisable to use this approach as it will require a lot of computation. However, the final result will be very close to the cheapest result. In terms of computation, it will be O(n), given that the number of passes the algorithm does is of a factor of a constant. Even though the result may be slightly worse than the brute force method, it will still be close to the cheapest result.